Service Function Chaining Simplified

Milad Ghaznavi, Nashid Shahriar, Reaz Ahmed, Raouf Boutaba

David R. Cheriton School of Computer Science, University of Waterloo, ON, Canada

{eghaznav | nshahria | r5ahmed | rboutaba}@uwaterloo.ca

Abstract— **Middleboxes have become a vital part of modern networks by providing *service functions* such as content filtering, load balancing and optimization of network traffic. An ordered sequence of middleboxes composing a logical service is called *service chain*. Service Function Chaining (SFC) enables us to define these service chains. Recent optimization models of SFCs assume that the functionality of a middlebox is provided by a single software appliance, commonly known as *Virtual Network Function* (VNF). This assumption limits SFCs to the throughput of an individual VNF and resources of a physical machine hosting the *VNF instance*. Moreover, typical service providers offer VNFs with heterogeneous throughput and resource configurations. Thus, deploying a service chain with custom throughput can become a tedious process of stitching heterogeneous VNF instances. In this paper, we describe how we can overcome these limitations without worrying about underlying VNF configurations and resource constraints. This prospect is achieved by distributed deploying multiple VNF instances providing the functionality of a middlebox and modeling the optimal deployment of a service chain as a mixed integer programming problem. The proposed model optimizes host and bandwidth resources allocation, and determines the optimal placement of VNF instances, while balancing workload and routing traffic among these VNF instances. We show that this problem is NP-Hard and propose a heuristic solution called *Kariz*. Kariz utilizes a tuning parameter to control the trade-off between speed and accuracy of the solution. Finally, our solution is evaluated using simulations in data-center networks.**

I. INTRODUCTION

Network Function Virtualization (NFV) is expected to instigate a revolutionary change in the networking industry. This industry still has the "mainframe" mindset relying on vendor specific, proprietary middleboxes providing various network functions. Examples of such middleboxes include firewalls, proxies, WAN optimizers, Intrusion Detection Systems (IDSs), etc. NFV proposes to replace proprietary, hardware middleboxes with innovative and flexible software middleboxes also known as Virtual Network Functions (VNFs).

VNFs are generally run on commodity (e.g., x86 based systems) hardware. In this way, the capital and operational expenditures of buying and maintaining specialized hardware is reduced. However, VNFs are yet to achieve the same performance of their hardware counterparts. This impedes the real life adoption of VNFs in today's networks carrying voluminous data traffic every second. In these networks, traffic is often required to pass through and processed by an ordered sequence of VNFs called *service chain*. For instance, traffic may need to pass through an IDS, then a proxy, and finally through a firewall. This phenomenon is commonly referred to as *Service Function Chaining* (SFC) [30]. Service chains or simply chains are required to process large volumes of traffic within a very short period of time to facilitate real-time streaming applications that comprise majority of traffic

in today's networks. Failure to provide the desired throughput of a chain may lead to violation of the service level agreements incurring high penalties. Hence, achieving high throughput of VNFs is of paramount importance.

There are several streams of on going researches towards increasing the throughput of a VNF. The first stream explores the possibility to build virtual platforms capable of processing packets very fast by utilizing advanced hardware technologies [18], [26]. The second stream combines VNFs with hardware middleboxes to facilitate a better usability of the existing hardware middleboxes [27] and brings the benefits of the both worlds. However, none of these approaches can overcome the physical limitation of deploying a VNF on a single physical machine. Hence, the third stream of works including [14], [31] propose to redistribute the traffic destined to a VNF across multiple VNF instances running independently on different CPU cores of a server, or even different servers altogether providing the functionality of the VNF. The cluster architecture of Bro IDS [29] is an example of such distributed deployment. In addition to achieving higher throughput, the distributed deployment offers better flexibility and reliability of the deployed chains than the standalone counterpart.

A fundamental problem for deploying a chain with a custom throughput is the resource efficient selection and placement of VNF instances. Solving this problem requires addressing several optimization challenges. First, there can be heterogeneity in terms of the throughput of different VNF instances. For instance, virtual WAN optimizer such as Riverbed STEELHEAD instances [5] have throughput of 10 and 50 Mbps whereas virtual firewalls such as Baracuda firewalls [1] have throughput of 100, 200, 400, and 750 Mbps. Hence, to attain a desired throughput in an chain of WAN optimizer and firewall, one has to enumerate all possible combinations of VNF instances for each of the VNFs and choose the combination minimizing the demand on physical resources (e.g., CPU cores, memory, etc). Furthermore, the chosen combination of VNF instances has to be placed into the physical machines/hosts in such a way that optimizes the overall bandwidth consumption of the chain. for example, placing a VNF instance far apart from other VNF instances of the same chain will result in increased bandwidth allocation along the path.

These problems are interdependent, and an optimal chain deployment has to solve them all together resulting in a joint optimization problem. Furthermore, a deployment solution should adhere the system implementation aspects regarding distributed deployment of VNF instances, traffic splitting, and accurate load distribution among these instances. Existing optimization models including [7] and [27] assume that the functionality of a middlebox is provided by a single VNF and have not studied this joint optimization problem. In this

paper, we address this joint optimization problem by taking into account the system implementation aspects. Specifically, our contribution in this paper are as follows:

- We develop an optimization model to deploy a chain in a distributed and resource efficient manner. Our proposed model abstracts heterogeneity of VNF instances and allows us to deploy a chain with custom throughput without worrying about individual VNF's throughput.
- We implement this model using Mixed Integer Programming (MIP) in CPLEX for finding optimal solutions in small scale networks.
- For larger scale networks, we propose *Kariz*, a local search heuristic, that employs a tuning parameter to balance the speed-accuracy trade-off.
- We evaluate Kariz compared to MIP implementation for various chain-lengths and throughput-demands. The results suggest that Kariz achieves the competitive acceptance ratio of $\sim 80\text{-}100\%$ at an extra cost of less than 25% in comparison to MIP model.

The rest of the paper is organized as follows. In Section II, we study the related work. Section III discusses the system implementation and deployment challenges. We present our problem formulation in Section IV. Our solution is proposed and evaluated in Section V and Section VI, respectively. Lastly, Section VII concludes this paper.

II. RELATED WORK

SFC deals with deployment of VNFs that are chained together to provide a collection of services. CoMb [34], for example, proposes a simplified VNF placement by putting all the VNFs dealing with the same flow on the same fixed physical node (called CoMb box). In contrast, our solution does not restrict VNFs to run on a fixed set of physical nodes, and can be deployed anywhere in the infrastructure.

Bari et al. model a batch deployment of chains, called VNF Orchestration Problem (VNF-OP) [7]. VNF-OP deploys each middlebox in one physical node. VNF-P [27] studies a hybrid scenario of hardware-middlebox and VNFs to provide requested service. None of these models assume that a middlebox is deployed in a distributed manner. Clayman et al. [10] consider the placement of VNFs with respect to several goals, including reducing energy consumption and load balancing. Based on these goals, the best performing algorithm out of *least used host*, *N at a time in a host*, and *least busy host* is chosen.

Sahhaf et al. propose to decompose a chain into more elementary and implementation-close components [32]. A selection mechanism determines a decomposition to minimize the mapping cost, and an algorithm deploys the selected decomposition. While this work focuses on the functional decomposition, our goal is to decompose the chain based on performance requirement. In addition, their algorithm does not consider the joint optimization properties of the problem.

The distributed deployment of a chain raises several challenging implementation questions including control plane design, VNF state management, and system abstraction. These

Table I: Comparison of Our Work to the Most Related Works

Paper	Methodology	Distributed VNF deployment	Traffic splitting	Accurate load distribution	Optimal deployment
VNF-OP [7]	Optimization	✗	✗	✗	✓
Service Dec. [32]	Optimization	✗	✗	✗	✓
Split/Merge [31]	System Imp.	✓	✓	✗	✗
OpenNF [14]	System Imp.	✓	✓	✗	✗
Our work	Optimization	✓	✓	✓	✓

challenges have addressed by [12], [14], [31]. Split/Merge [31] proposes a system to address challenges of VNF state management and traffic route management. Stratos [12] uses a rather simple technique for both the initial and subsequent placement of VNFs. It packs VNFs that belong to the same chain as close as possible. OpenNF [14] supports the idea of packet processing to be redistributed across a collection of VNF instances. However, its focus is to provide a coordinated control plane framework for both internal VNF state and network forwarding state. As such, none of these works consider the optimization problem of deploying a VNF chain in a distributed fashion.

The related works discussed above are compared in Table I in view of four important aspects of the distributed VNF orchestration problem. From the comparison, it is apparent that none of the existing works have considered all the aspects of the optimization problem we study in this paper.

III. CHALLENGES

A service chain specifies that the traffic originating from a *source*, is processed by an ordered sequence of middleboxes, and finally is delivered to a *target*. To have transparent underlying VNF instances as well as abstracting the resource requirements of these instances, several system implementation and optimization challenges have to be addressed.

A. System Implementation Challenges

Middleboxes often operate on data-packets in a *flow* granularity and maintain *state information* on the flows and sessions they process [36], [38]. The state information consists of configuration and statistical data, and differs from one middlebox to another. By replacing a middlebox with multiple *VNF instances*, the functionality should not change, and these instances have to act unified. Moreover, the traffic processed by a single middlebox, now should be processed by multiple VNF instances. Thus, *consistent state distribution* and *consistent traffic distribution* among the VNF instances are essential.

1) Consistent State Distribution: Deployment of multiple VNF instances to provide functionality of a middlebox requires distribution of the state information. Hence, we need to *model* the state information of middleboxes and *distribute* the state information among the VNF instances consistently. The state information can be classified as *internal* and *external*. The internal state is only stored and used by a single instance, while the external state is distributed and shared across multiple instances. Since the state information is stored in a key-value structure [19], [33], [36], data structures like distributed hash-tables and technologies like Remote Direct Memory Access (RDMA) can fulfill this challenge efficiently. Moreover, it might require to modify the middleboxes to cope with the defined model. There are abstraction models

and system implementations that address this challenge. Rajagopalan et al. [31] introduce a system-level abstraction called Split/Merge that store the internal state exclusively inside each VNF instance, while the external state is distributed and accessible among other instances. As a proof of concept, they implemented FreeFlow as a Split/Merge system, and ported Bro IDS [29] inside it. Further, they analyzed and confirmed the compatability of two other middleboxes, i.e. application delivery controller and stateful NAT64. In addition, Joseph and Stoica [19] provides a model to describe different middleboxes. As concrete examples, firewall, NAT and layer4 and layer 7 load balancer are described using the proposed model. Moreover, Qazi et al. [13] and OpenNF [14] introduce a unified framework to manage the state information.

2) Consistent Traffic Distribution: By replacing a single middlebox with multiple VNF instances, *splitting* and *balancing* the traffic load among these instances are necessary. Per-flow traffic splitting distributes the traffic in granularity of flows, and packets of a flow have to be routed along the same path. Split/Merge [31] utilizes a similar approach. However, this approach does not support accurate load distribution and is not always applicable. For instance, if the load of a flow is higher than the throughput of assigned VNF instance, it cannot handle the load and we have to split the traffic to a smaller granularity. *Flowlet switching* [8], [20], [35] can be leveraged to split the traffic in a more fine-grained granularity. A flowlet is a "burst of packets from the same flow followed by an idle interval" [35]. If the interval between two flowlets is greater than the maximum difference of parallel paths, the second flowlet –and consequaently following flowlets– can be sent through different paths. Thus, a single flow can split into multiple paths without packet-reordering. Furthermore, accurate load balancing is achieved using short flowlet intervals ($[50, 100]ms$) [35]. Specifically, flowlets are abundant in data center networks since the latency is very low and the traffic is intensively bursty [21]. In addition to these distributed methods, the central schemes leveraging SDN and OpenFlow capabilities [23] can also be used. For instance, *group tables* [4] can be used to split and balance the traffic. Combining these schemes with virtualization technologies, such as VXLAN [24] and NVGRE [11] can provide consistent traffic distribution for deployed chains.

We showed the feasibility of distributed deployment of VNF instances to provide the functionality of a middlebox and distributing traffic among these instances. Here, we clearly mention our assumptions to build the ground for our optimization model.

- The state information of middleboxes can be classified and distributed among multiple VNF instances.
- VNF instances of the same middlebox act as a single unit by accessing the distributed state information.
- The host resource overhead of accessing distributed state information is considered in resource demands of VNFs.
- Multi-path routing of a single flow among the VNF instances does not alter the functionality of instances as a whole, and shared distributed state information is

Table II: VNFs

Middlebox	VNF	Throughput	CPU demand	Memory demand
IDS	IDS_1	50 Mbps	1 core	24 GB
	IDS_2	80 Mbps	1 core	32 GB
Firewall	FW_1	100 Mbps	1 core	1.75 GB
	FW_2	200 Mbps	2 core	3.50 GB

sufficient for the correct functionality.

- The communication overhead to access the distributed state information is negligible compared to the actual service traffic volume.
- VNF instances belonging to the same middlebox process the same amount of traffic in similar amount of time.

B. Optimization Challenges

The optimization challenge is computing an optimal allocation of *host* and *bandwidth* resources to a chain. For each middlebox in a chain, a number of *instances* of each *VNF* are placed to provide the requested throughput. These instances are placed in a set of selected hosts. In addition, the traffic is split and routed among the placed instances. Therefore, following decisions have to be made optimally: *Number of instances* of each VNF, *placement* of these instances in a set of hosts, and *routing the traffic* among the placed instances. These decisions are dependent and need to be made together.

Fig. 1 depicts a deployment of a chain. The substrate network of Fig. 1a consists of 6 hosts. Each host has 8 core CPU and 64 GB residual memory. For the sake of simplicity in this example, the switches are not shown, and we assume that presented substrate paths are disjoint. All substrate paths have 130 mbps available bandwidth. The chain of Fig. 1b consists of two VNFs with 210 mbps throughput: an Intrusion Detection System (IDS) and a firewall (FW). The traffic flow comes from host A, the source, and after being processed by IDS and FW is sent to host F, the target. As listed in Table II, there are 4 VNF types for IDS and FW. Fig. 1c depict the deployed service chain in the network, and Fig. 1d shows the logical representation of this deployment. As shown, three instances for IDS (one IDS_1 and two IDS_2) and two instances for FW (one FW_1 and one FW_2) are placed. The IDS instances are installed in hosts B and D. The traffic flow splits, and 80 mbps and 130mpbs is routed from the source to hosts B and D, respectively. FW instances are installed in hosts B and E. In host B, the traffic flow after being processed by IDS_2 is sent to FW_1. Furthermore, IDS_1 and IDS_2 forward the traffic flow to host C in which instance FW_2 is placed. Finally, the traffic flow from the FW instances is sent to the target. Note that it is possible to place the VNF instances in the source and target if there are sufficient available host resources.

IV. SERVICE FUNCTION CHAINING SIMPLIFIED

Having the assumptions established and optimization challenges discussed, we introduce the formal definitions followed by the mathematical model.

A. Definitions

1) Physical Resources: $R = \{$CPU, memory, storage, $\dots\}$ represents a set of available physical resources.
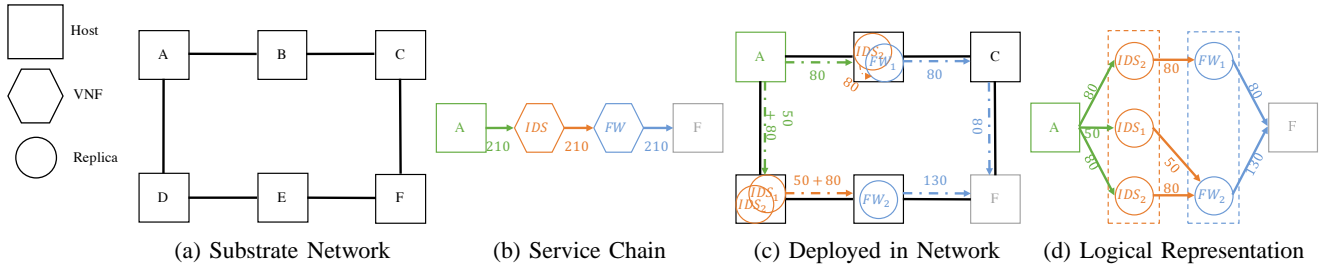
Figure 1: Deployment of a Service Chain

2) Substrate Network: Graph $G = (N, E)$ is the substrate network, where N and E are substrate nodes and links, respectively. We use index notation for substrate nodes. For instance, $m < n$ for nodes $m, n \in N$ means that index of m is less than index of n. Let $c_{mr} \in \mathbb{R}^+$ denotes the residual capacity of node $m \in N$ for resource $r \in R$. Set $E_m \subseteq E$ represents incident links on node m. Moreover, $(m, n) \in E$ is the link between node $m \in N$ and node $n \in N$ and has residual bandwidth capacity of $c_{mn} \in \mathbb{R}^+$.

3) Service Chain: Forwarding graph $\overline{G} = (\overline{N}, \overline{A})$ denotes a chain. We use Service Function (SF) and middlebox synonymously. $\overline{N}$ includes SFs $\overline{V} \subset \overline{N}$, and two endpoints $\overline{s}$ and $\overline{t}$. Traffic flow coming from $\overline{s} \in \overline{N}$, is processed by SFs in the chain, and is forwarded to $\overline{t} \in \overline{N}$. $\overline{s}$ and $\overline{t}$ are the *source* and *target* of the traffic, respectively. Corresponding substrate nodes for source and target are respectively $s \in N$ and $t \in N$. SF $\overline{v} = f(\overline{u})$ is following SF next to SF $\overline{u}$. We define *ring* $(\overline{u}, \overline{v}) \in \overline{A}$ as two consecutive SFs $\overline{u}, \overline{v} \in \overline{N}$, where $\overline{v} = f(\overline{u})$. We assume that $\overline{u}$ generates traffic of type $\overline{u}$ and $\overline{v}$ consumes this traffic type. Each ring $(\overline{u}, \overline{v}) \in \overline{A}$ has the *throughput demand* of $\overline{b}$ representing the integer volume of traffic flow that is generated or consumed by the ring nodes.

4) VNFs: Set V denotes VNFs. Each VNF $u \in V$ has throughput $q_u \in \mathbb{R}^+$ showing the maximum traffic volume that u can process. Besides, $d_{ur} \in \mathbb{R}^+$ is the demand of u for resource $r \in R$. For $\overline{s}, \overline{t} \in \overline{N}$, we assume there are VNFs $u_{\overline{s}} \in \overline{V}$ and $u_{\overline{t}} \in \overline{V}$, respectively. These VNFs have throughput of $\overline{b}$ and no demand for any resource. Finally, VNFs of type $\overline{u} \in \overline{V}$ are identified by $V_{\overline{u}}$.

B. Mathematical Model

1) Decision Variables: $x^{\overline{u}}_{mn} \in \mathbb{R}$ is the volume of traffic of type $\overline{u} \in \overline{N}/\{\overline{t}\}$ on substrate link $(m, n) \in E$. Target $\overline{t}$ is excluded from this definition because it only consumes the traffic, therefore no traffic of this type exists in the network. Variable $y_{mu} \in \mathbb{Z}$ is the number of instances of VNF $u \in V$ in substrate node $m \in N$. VNF instances of $V_{\overline{u}}$ installed in node $m \in N$ provide throughput of type $\overline{u} \in \overline{N}/\{t\}$. Decision variable $z_{m\overline{u}} \in \mathbb{R}$ denotes the allocated throughput of these VNF instances. A solution for the problem is represented by a tuple of allocation vectors (X, Y, Z) which are defined as follows. Let vector $X_{\overline{u}} = \{x^{\overline{u}}_{mn} : \forall (m, n) \in E\}$ be allocated bandwidth of links to traffic of type $\overline{u}$, and $X = \bigcup_{\overline{u} \in \overline{N}/\{t\}} X_{\overline{u}}$. If $Y_{\overline{u}} = \{y_{mu} : \forall m \in N, \forall u \in V_{\overline{u}}\}$ identifies the VNF instantiated for SF $\overline{u} \in \overline{N}$, let $Y = \bigcup_{\overline{u} \in \overline{N}/\{t\}} Y_{\overline{u}}$. Finally,

$Z_{\overline{u}} = \{z_{m\overline{u}} : \forall m \in N\}$ denotes allocated throughput of type $\overline{u} \in \overline{N}/\{t\}$ in every node, and $Z = \bigcup_{\overline{u} \in \overline{N}/\{t\}} Z_{\overline{u}}$.

2) Substrate Node Capacity Constraint: Eq. 1 guarantees the resource capacities of substrate nodes in which instances are placed are respected.

$$\forall m \in N : \forall r \in R : \sum_{u \in V} y_{mu} d_{ur} \leq c_{mr} \qquad (1)$$

3) Location Constraint: Equalities in Eq. 2 ensure that a instance of $u_{\overline{s}}$ and a instance of $u_{\overline{t}}$ are only placed in $s \in N$ and $t \in N$, respectively.

$$y_{su_{\overline{s}}} = 1, \quad \sum_{m \in N/\{s\}} y_{mu_{\overline{s}}} = 0$$
$$y_{tu_{\overline{t}}} = 1, \quad \sum_{m \in N/\{t\}} y_{mu_{\overline{t}}} = 0 \qquad (2)$$

4) Substrate Link Capacity Constraint: Eq. 3 makes sure that the capacities of substrate links are not violated.

$$\forall (m, n) \in E, m < n : \sum_{\overline{u} \in \overline{N}} (x^{\overline{u}}_{mn} + x^{\overline{u}}_{nm}) \leq c_{mn} \qquad (3)$$

5) Throughput Constraint: Eq. 4 assures that the aggregate throughput capacity of instances of VNFs of type $\overline{u} \in \overline{N}$ placed in substrate node $m \in N$ is more than allocated throughput $z_{m\overline{u}}$.

$$\forall m \in N : \forall \overline{u} \in \overline{N} : \sum_{u \in V_{\overline{u}}} y_{mu} q_u \geq z_{m\overline{u}} \qquad (4)$$

6) Throughput Demand Constraint: Eq. 5 guarantees that for each SF $\overline{u} \in \overline{V}$, throughput of $\overline{b}$ is allocated by VNF instances of $V_{\overline{u}}$.

$$\forall \overline{u} \in \overline{N} : \sum_{m \in N} z_{m\overline{u}} = \overline{b} \qquad (5)$$

7) Flow Conservation Constraint: Eq. 6 is the modified version of flow conservation constraint [37]. Let say in node $m \in N$, VNF instances of types $\overline{u}$ and $\overline{v} = f(\overline{u})$ are installed. Therefore, VNF instances of $V_{\overline{v}}$ locally process a volume of traffic of type $\overline{u}$ generated by instances of $V_{\overline{u}}$. This volume is $z_{m\overline{v}}$. Not processed traffic volume should comes outside the node m. This constraint assures this phenomenon.

$$\forall m \in N : \forall \overline{u} \in \overline{N}/\{\overline{t}\} : \overline{v} = f(\overline{u}) :$$
$$\sum_{(m,n) \in E_m} (x^{\overline{u}}_{mn} - x^{\overline{u}}_{nm}) = (z_{m\overline{u}} - z_{m\overline{v}}) \qquad (6)$$

8) Bandwidth Allocation Cost: Eq. 7 denotes the bandwidth resource allocation cost. Coefficient $\beta \in \mathbb{R}^+$ identifies the relative importance of bandwidth resources. Analogously, $B(X_{\overline{u}})$

is the bandwidth cost for SF $\overline{u}$.

$$B(X) = \sum_{\overline{u} \in \overline{N}/\{t\}} \sum_{(m,n) \in E} \beta x_{mn}^{\overline{u}} \qquad (7)$$

9) Host Resource Allocation Cost: Eq. 8 is the cost of allocating host resources to place VNF instances. $\alpha_r \in \mathbb{R}^+$ is a coefficient denoting the relative importance of resource $r \in R$. Similarly, $H(Y_{\overline{u}})$ and $H(y_{mu})$ represent this cost for SF $\overline{u} \in \overline{V}$ and VNF $u \in V$, respectively. Note that we can compute cost of $H(y_{mu})$ if $z_{m\overline{u}}$ is given[1]. Let $H(z_{m\overline{u}})$ be this computed cost.

$$H(Y) = \sum_{u \in V} \sum_{r \in R} \alpha_r d_{ur} y_{mu} \qquad (8)$$

10) Objective Function: Eq. 9 is minimization of aggregate cost of allocating host and bandwidth resources.

$$\min \Big(B(X) + H(Y) \Big) \qquad (9)$$

This problem is NP-Hard. Even if the number of instances and throughput allocations for every VNF are known, the problem still generalizes the NP-Hard problem of *virtual network embedding problem with path splitting* [9], [39]. Due to intractability of the problem for larger scales, we introduce a heuristic which approximates the optimal solution in a reasonable time.

V. KARIZ: HEURISTIC SOLUTION

Before explaining our solution, we construct a visualization tool to simplify our description. Let assume that each $\overline{u} \in \overline{N}$ is deployed in a *layer*. Each layer contains a set of nodes in which VNF instances of corresponding type can be installed. In other words, in the layer corresponding to $\overline{u}$, we initially place nodes in which at least a VNF $v \in V_{\overline{u}}$ can be instantiated. More precisely, this layer is a subset of nodes and is denoted by $L(\overline{u})$. Fig. 2c depicts the layers for chain. As shown in Fig. 2c, s and t are the only present nodes in layers $L(\overline{s})$ and $L(\overline{t})$, respectively. Further, nodes $\{s, m\}$ and $\{n, t\}$ are respectively included in layers $L(\overline{u})$ and $L(\overline{v})$ because these nodes have sufficient resource to host VNF instances of these SFs. We can now describe our problem as the problem of routing between layers to bring the traffic from the first layer $L(\overline{s})$ to last layer $L(\overline{t})$. In each layer $L(\overline{u})$, traffic passes through a set of nodes in which VNF instances of $V_{\overline{u}}$ are placed. Fig. 2d presents a sample solution for the chain of Fig. 2b.

Inspired by [17], [28], we develop a local search heuristic, *Kariz*, which routes traffic layer by layer. We provide the process first, and then explain an overview of the details. Kariz is shown in Alg. 1 and works as follows. At the beginning, we set initial solution as empty (line 1). Starting from layer $L(\overline{s})$ (line 2), iteratively route $\overline{b}$ volume of traffic from layer $S = L(\overline{u})$, *source-layer*, to next layer $T = L(\overline{v})$, *sink-layer* (lines 3-11). After finding the optimal route between two layers (line 5), compute the number of VNF instances of $V_{\overline{v}}$ by considering the allocated throughput (line 6). Add the solution of sink-layer to the earlier solution (line 7). Improve

[1]By solving a variant of knapsack problem as explained in Section V-A

(a) Substrate Network   (b) Service Chain

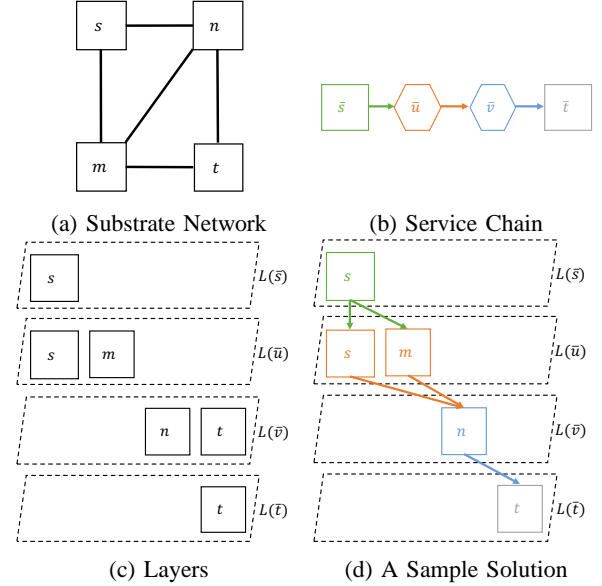(c) Layers   (d) A Sample Solution

Figure 2: Layers

the current solution (line 8), and update layers (line 9). Now, traffic has reached the sink-layer; consider this layer as new source-layer (line 10). Repeat this procedure if traffic has not reached the last layer yet, and there are nodes in new source-layer (line 11).

Algorithm 1 Kariz Algorithm

1: $(X, Y, Z) \leftarrow (\emptyset, \emptyset, \emptyset)$;
2: $\overline{u} \leftarrow \overline{s}$; $z_{s\overline{s}} \leftarrow \overline{b}$; $z_{t\overline{t}} \leftarrow \overline{b}$; $S \leftarrow L(\overline{s})$;
3: **do**
4:      $\overline{v} \leftarrow f(\overline{u})$; $T \leftarrow L(\overline{v})$;
5:      $X_{\overline{v}}, Z_{\overline{v}} \leftarrow route(S, T, \overline{b})$;
6:      $Y_{\overline{v}} \leftarrow vnf\text{-}instances(Z_{\overline{v}})$;
7:      $(X, Y, Z) \leftarrow (X \cup X_{\overline{v}}, Y \cup Y_{\overline{v}}, Z \cup Z_{\overline{v}})$;
8:      $improve(X, Y, Z)$;
9:      $update\text{-}layers(Y)$;
10:      $\overline{u} \leftarrow \overline{v}$; $S \leftarrow L(\overline{v})$;
11: **while** $(\overline{u} \neq t$ and $S \neq \emptyset)$;

Yet, we have not clarified how the routing between two layers and the number of VNF instances in the sink-layer are computed; how the solution is improved; also how the layers are updated.

A. Route and VNF Instances

Function $route(.)$ in Alg. 1 computes the route between two layers by solving the multi-source multi-sink Minimum Cost Flow Problem (MCFP) [16]. MCFP is the problem of routing a volume (say $\overline{b}$) of a *commodity* (in our case traffic of type $\overline{u}$) from multiple sources (say source-layer) to multiple sinks (in our case sink-layer). Any multi-source multi-sink MCFP can be modeled as a single-source single-sink MCFP which is solvable in polynomial time [16]. For our problem, this is achieved by representing the source- and sink-layers with imaginary nodes *super-source* and *super-sink*, respectively. Fig. 3 depicts this model for layers S and T in Fig. 2. The procedure is as follows. Add super-source and connect it to
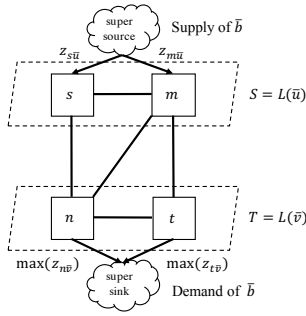
Figure 3: Routing as Single-Source Single-Sink MCFP

every node $m \in S$ in the source-layer with a directed-link whose capacity is $z_{s\overline{u}} \in Z$. For the sink-layer, add super-sink node and connect every node $n \in T$ using a directed-link. The capacity of the directed-link connecting node n to super-sink is the maximum throughput $\max(z_{n\overline{v}})$ of the VNF instances that can be installed in node n. There is no cost to send the traffic via these links. As the result, the minimum cost route of traffic from super-source to super-sink gives the optimal routing between the two layers. If p represents super-sink, the throughput allocation in each $n \in L(\overline{v})$ is $z_{n\overline{v}} = x_{np}^{\overline{u}}$.

Finding the capacity of directed-links from sink-layer to the super-sink is similar to the problem of function $vnf\text{-}instances(.)$. Former is finding the maximum throughput $\max(z_{n\overline{v}})$ out of VNF instances that can be installed in node n. Latter is finding the minimum allocation of resources to VNF instances providing throughput of at least $z_{n\overline{v}}$ in each node $n \in L(\overline{v})$. In fact, these two problems are *dual* and can be modeled as a *multidimensional knapsack problem* [22]. Think of the node as $|R|$-*dimensional knapsack*, each *dimension* corresponding to a resource $r \in R$. The *items* to be packed are VNF instances with *profits* of their throughputs and *weights* of their host resources demands. Although this problem is known to be NP-Hard [22], since the resources of a single physical machine, specially number of CPU cores are limited, and the problem size is small. Hence, we can solve it efficiently. Alternatively, as CPU cores are the most expensive and restricted resources, a feasible solution optimizing the number of allocated cores is a good optimum.

B. Solution Improvement Rounds

Routing of traffic between two layers might result in fragmented host resource allocation with high cost. Therefore, we need to improve the solution. Function $improve(.)$ as presented in Alg. 2 facilitates this: Repeatedly search for some *actions* to improve the solution (lines 2-8). If no such action is found, report the current solution (line 4-6). Otherwise, perform the action with greatest drop in the cost, the best *admissible* action (line 7), and continue with the adjusted solution. We define actions and *admissibility* in Section V-B1 and Section V-B3, respectively.

1) Actions: An action is a *local transformation* intended to reduce the solution cost. Let (X', Y', Z') be the modified solution after performing an action on a current solution (X, Y, Z). The cost difference before and after performing an

---

Algorithm 2 Function $improve(.)$

1: **function** $improve(X, Y, Z)$
2:     **loop**
3:         $a \leftarrow best\text{-}action(X, Y, Z)$;
4:         **if** not $admissible(a)$ **then**
5:             return (X, Y, Z);
6:         **end if**
7:         $perform\text{-}action(X, Y, Z, a)$;
8:     **end loop**
9: **end function**

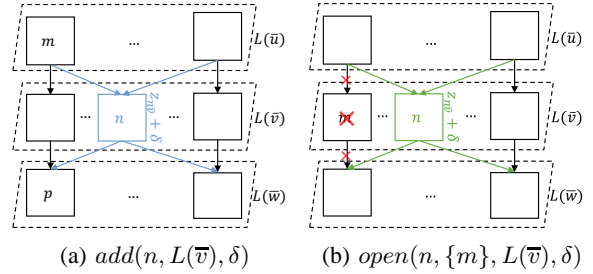(a) $add(n, L(\overline{v}), \delta)$ (b) $open(n, \{m\}, L(\overline{v}), \delta)$

Figure 4: Actions

action is regarded as the *action cost*, as defined in Eq. 10. The best action has the lowest cost.

$$\left(B(X') + H(Y') \right) - \left(B(X) + H(Y) \right) \qquad (10)$$

We define the following actions that are variants of actions used by [28]:

- $add(n, L(\overline{v}), \delta)$: Include node $n \in N$ in $L(\overline{v})$ and allocate more $\delta > 0$ units of throughput in this node ($z_{n\overline{v}} \leftarrow z_{n\overline{v}} + \delta$). Then, find the minimum cost routing from layer $L(\overline{v})$ to next and previous layers in the current solution, given allocated throughputs of $L(\overline{v})/\{n\}$. The next and previous layers are $L(\overline{w})$ and $L(\overline{u})$ if $\overline{w} = f(\overline{v})$ and $\overline{v} = f(\overline{u})$, respectively. Finally, tune the allocated throughput of nodes $L(\overline{v})$. This action is shown in Fig. 4a.
- $open(n, M, L(\overline{v}), \delta)$: Add node $n \in N$ into layer $L(\overline{v})$, remove nodes $M \subseteq L(\overline{v})$, and allocate more $\delta > 0$ units of throughput in node n ($z_{n\overline{v}} \leftarrow z_{n\overline{v}} + \delta$). Finally, reroute the traffic either received or originated in layer $L(\overline{v})$. This action replaces a set of fragmented VNFs installed in different nodes M with VNFs collocated in one node n. This action makes sense only if $\delta \geq \sum_{m \in M}(z_{m\overline{v}})$. We used a similar action, $install$, in elastic placement of VNFs in [15]. Fig. 4b depicts this action.

Traffic routing in the above actions is a bit different from routing in function $route(.)$. The difference is routing of two different traffic types. Still this problem is tractable, and we can model it as a multi-commodity MCFP that is solvable in polynomial time.

We also need to examine actions and select the best in polynomial time and ensure that the number of performed actions is not exponential. Particularly, we need to select the best action with sufficient improvement efficiently. These criteria, *efficient action selection* and *sufficient improvement*, are essential to assure that the algorithm terminates in polynomial time.

2) Efficient Action Selection: The number of possible $add(n, L(\overline{v}), \delta)$ actions are at most $|N| \times |\overline{V}| \times \overline{b}$ under the assumption of integrality of $\overline{b}$. Hence, it is possible to check all actions and select the best one in polynomial time. We can even do better and select the value of δ by considering the throughputs of VNFs $V_{\overline{v}}$. However, number of possible $open(., M, L(\overline{v}), .)$ actions can be exponential because of the large number of possible subsets $M \subseteq L(\overline{v})$. Thus, we need an efficient procedure to select a good $open(.)$ action. For a fixed layer $L(\overline{v})$, fixed node $n \in N$ and fixed δ, we find this subset in a greedy procedure working as follows. Starting from empty set M, iteratively remove a node m from $L(\overline{v})$ and add it to M. Removing this node has the minimum cost vs. other nodes $L(\overline{v})/m$. Continue this procedure while such a node $m \in L(\overline{v})$ exists, the removal of m decreases the cost, and m's throughput is less than $\delta - \sum_{m \in M} z_{m\overline{v}}$. This procedure repeatedly removes an individual node $m \in L(\overline{v})$ whose removal produces the highest decrease in both bandwidth and host resource allocation costs.

3) Sufficient Improvement: Still the number of actions can be large due to exponential number of performed actions with minor improvement. To solve this problem, only actions with sufficient improvement of the cost are applied. An action with sufficient improvement is called *admissible*. More precisely, we define an action as admissible if it improves the solution no less than $\frac{\epsilon}{4|N|}\big(B(X) + H(Y)\big)$ for some tuning parameter $\epsilon > 0$ [25]. Using ϵ, we can control the trade-off between accuracy and speed of our solution. Let (X^*, Y^*, Z^*) be the optimal solution. Since the optimal solution is the lower bound for our solution, the number of performed actions will be at most $\frac{4|N|}{\epsilon} \ln \frac{B(X)+H(Y)}{B(X^*)+H(Y^*)}$.

C. Update Layers

As the last piece of the puzzle, function $update\text{-}layers(.)$ updates the nodes in every layer. From a layer $L(\overline{u})$ to which traffic is already reached, every node $m \in L(\overline{u})$ is eliminated if this node does not allocate throughput of type $\overline{u}$. From other layers, nodes whose resources are allocated and hereafter cannot host corresponding VNF instances are excluded. Layers $L(\overline{s})$ and $L(\overline{t})$ are kept out of the update.

VI. EVALUATION

A. Experimental Setup

1) Simulated Network: 6-ary Fat-tree [6], a common data-center topology, is used as the simulated network containing 99 nodes (54 hosts and 45 switches) and 162 links providing full bisection bandwidth. Hosts are equipped with 8 core CPU and 1 Gbps network adapter. The link capacities are 1 Gbps. The relative importance of allocating 1 Mbps of bandwidth over one link vs. one core CPU is 1%.

2) VNFs: We select firewall, IDS, IPsec and WAN-opt. as SFs. Table III reveals the VNFs used in the simulation. Since CPU is the most restricted host resource while dominating the cost, we ignore memory and storage requirements.

Table III: Off-the-shelf VNFs

Middlebox	VNF	Throughput	CPU demand
Firewall [1]	Level 1	100 Mbps	1 core
	Level 5	200 Mbps	2 core
	Level 10	400 Mbps	4 core
IDS	Bro [2]	80 Mbps	1 core
IPSec [3]	VSR1001	268 Mbps	1 core
	VSR1004	580 Mbps	4 core
WAN-opt. [5]	CCX770M	10 Mbps	2 core
	CCX1555M	50 Mbps	4 core

3) Service Chains: Sources and targets are uniformly distributed in the data-center network. Poisson distribution with the average of 1-chain per 100-seconds is used to simulate the arrival rate. Chains lifetimes follow the exponential distribution with an average of 3 hours.

4) Parameters: We asses Kariz in respect to *throughput-demand* and *length* of chains. In each experiment, the throughput-demand is fixed to one of $\{100,150,200,250,300\}$ Mbps, and one of the following chains is selected.

- Len-1: {Firewall},
- Len-2: {Firewall → IDS},
- Len-3: {Firewall → IDS → IPSec}, and
- Len-4: {Firewall → IDS → IPSec → WAN-opt.}

Note that Len-i contains all SFs of Len-i-1. We consider Len-1 and Len-2 as homogeneous chains because firewall and IDS VNFs in Len-2 almost demand the same resources for the same throughput. Len-3 and Len-4 are more heterogeneous due to different resource requirements of corresponding VNFs.

5) Evaluation Method: We compare Kariz against the optimal solution implemented using CPLEX. We refer to the optimal solution by *MIP*. The tuning parameter of Kariz is set to $\epsilon = 20$. Thus, an action is performed if it improves the current solution by 5%. With fixed parameters, we repeat each experiment 10 times for different generated 1000 chains, and report the arithmetic mean. In comparison charts, the ratio of Kariz's to MIP's corresponding value is reported.

B. Acceptance Ratio

The acceptance ratio results are shown in Fig. 5. Fig. 5a and Fig. 5b depicts the acceptance ratio of Kariz and MIP, respectively. The values are the average of acceptance ratios of 10 experiments. As expected, the longer chains with higher throughput-demand have the less chance to be accepted. The low acceptance ratio for Len-4 is due to resource hungriness of these chains, especially for WAN-opt. VNFs.

The range of number of accepted chains by Kariz vs. MIP in Fig. 5c are as follows: 95-100% for Len-1, 82-95% for Len-2, 79-100% for Len-3, and 89-102% for Len-4. Note that higher acceptance ratio for Kariz makes sense. Consider a situation that MIP accepts a hard to deploy chain rejected by Kariz. MIP allocates the resources, not allocated by Kariz. Consequently, this allocation prevents MIP from accepting some of the next chains; despite that, Kariz assigns not-allocated resources to these chains resulting in higher acceptance-ratio.

Considering chain length and throughput-demand impacts in Fig. 5c, Kariz performs closely to MIP. It might be expected that increasing the length of chain and throughput-demand should deteriorate Kariz's acceptance ratio vs. MIP. However, Kariz has better results for Len-3 and Len-4 than Len-2 and

Len-1, especially for 250 Mbps throughput-demand. Recall from Section V-B, Kariz attempts to improve the solution after deployment of every SF of a chain. Since, Len-4 and Len-3 include all SFs of Len-2 and Len-1 chains (see Section VI-A4), the expense of more improvement rounds increases the chance of adjusting the earlier solution. All in all, Kariz has a competitive acceptance ratio within 79-100% vs. MIP.

C. Resource Utilization

Resource utilization of Kariz is compared with MIP in Fig. 6. Bandwidth/CPU utilization for Kariz and MIP are the ratio of allocated bandwidth/CPU resources over aggregated bandwidth/CPU capacities in the network. Regarding VNF resources, the reports are the arithmetic mean of per-SF throughput utilization provided by placed VNF instances.

Bandwidth utilization ratios as depicted in Fig. 6a are: 97-101% for Len-1, 88-106% for Len-2, 78-111% for Len-3, and 101-131% for Len-4. Fig. 6a and Fig. 5c shows that Kariz efficiently utilizes the bandwidth resources for Len-1, Len-2, and Len-3 for various throughput-demands. Regarding Len-4, the efficiency of utilizing bandwidth resources is very close to MIP for throughput-demand of 100 and 200 Mbps. However, the efficiency of bandwidth utilization decreases for other throughput demands.

The CPU utilization ratios are in the range of 95-100% for Len-1, 84-95% for Len-2, 76-100% for Len-3, and 100-103%, as observed in Fig. 6b. According to Fig. 6b and Fig. 5c, Kariz utilizes the CPU resources in an efficient way close to MIP.

Finally, the VNF utilization ratios vs. MIP are shown in Fig. 6c. Following ranges are reported: 100-100% for Len-1, 99-100% for Len-2, 101-105% for Len-3, and 101-111%. Evidently Kariz utilizes VNF instances very closely to MIP for different lengths and throughput demands.

D. Operational Costs

Fig. 7 shows Kariz's costs vs MIP. We collect the Kariz's and MIP's average of per chain costs. The reported values are the ratio of Kariz's and MIP's costs. As shown in Fig. 7a on average, Kariz allocates bandwidth resource vs. MIP in the range of: 101-102% for Len-1, 105-111% for Len-2, 101-109% for Len-3, and 100-140% for Len-4. Regarding CPU as presented in Fig. 7b, on average the same number of CPU cores is allocated for Len-1 and Len-2. For Len-3, 0-3% less number of CPU cores are allocated. Also, 2-13% more number of CPU cores are allocated to Len-4 by Kariz. Finally, in respect to total operational cost in Fig. 7c, following cost ratios vs MIP are observed: 100-101% for Len-1, 103-107% for Len-2, 100-105% for Len-3, and 99-125% for Len-4. Note that it makes sense that Kariz pays 1% less cost than MIP per Len-4 chains. These solutions accept different number of chains in presence of different available resources. For instance, MIP might accept a chain when the resources are scarce, while Kariz not finding a feasible solution rejects this chain. Consequently, MIP would pay more operational cost in average. In summary, Kariz incurs competitive per-chain cost less than 125% of MIP.

VII. Conclusion

Recent optimization models of SFC assume that functionality of a middlebox is provided by a single VNF. This assumption limits SFC to either a single VNF or an individual physical machine. Moreover, heterogeneity of throughput and resource configurations of miscellaneous VNFs makes deployment of a service chain complex. In this paper, we described how we can overcome these limitations. We introduced a mathematical model that enables us to deploy multiple VNF instances to provide the functionality of a middlebox. This eliminates the throughput bound of a chain to a single VNF or a single physical machine. Moreover, this model abstracts heterogeneity of VNFs and allows us to define chains with custom throughput without worrying about individual VNF throughputs. In addition, our Mixed Integer Programming (MIP) model gives the optimal deployment of a chain. For larger scales, we proposed and evaluated a heuristic called Kariz. The experimental results for various chain lengths and throughput demands suggest that Kariz achieves a competitive acceptance ratio of $\sim$ 80-100% with an extra cost of less than 25% compared to MIP model.

References

[1] Barracuda WAF. https://www.barracuda.com/assets/docs/Datasheets/Barracuda_Web_Ap [Online].

[2] Bro. https://www.bro.org/sphinx/cluster/index.html. [Online].

[3] HP Virtual Router Series. http://www8.hp.com/us/en/products/networking-routers/produ

[4] Openflow switch specication v.1.3.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specificat

[5] Steelhead Product Family Spec Sheet. http://media-cms.riverbed.com/documents/Spec+Sheet+-+Steelhead+Family+-+05.06.2(

[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *ACM SIGCOMM 2008*.

[7] M. Al-Fares, A. Loukissas, and A. Vahdat. On Orchestrating Virtual Network Functions in NFV. In *IEEE CNSM 2015*.

[8] Mohammad Alizadeh et al. Conga: Distributed congestion-aware load balancing for datacenters. *SIGCOMM Comput. Commun. Rev.*

[9] N. Chowdhury, M. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*.

[10] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca. The dynamic placement of virtual network functions. In *IEEE NOMS, 2014*.

[11] Pankaj Garg and Yu-Shun Wang. Nvgre: Network virtualization using generic routing encapsulation. 2014.

[12] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella. Stratos: Virtual middleboxes as first-class entities. *UW-Madison TR1771*, 2012.

[13] Aaron Gember et al. Toward software-defined middlebox networking. In *11th ACM Workshop on Hot Topics in Networks*. ACM, 2012.

[14] Aaron Gember-Jacobson et al. Opennf: Enabling innovation in network function control. SIGCOMM 2014, 2014.

[15] Milad Ghaznavi et al. Elastic virtual network function placement. In *CloudNet 2015*, Oct 2015.

[16] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, October 1989.

[17] S. Guha et al. Hierarchical placement and network design problems. In *41st Annual Symposium on Foundations of Computer Science*, 2000.

[18] Jinho Hwang et al. Netvm: high performance and flexible networking using virtualization on commodity platforms. *Network and Service Management*, 2015.

[19] D. Joseph and I. Stoica. Modeling middleboxes. *Network, IEEE*, 22(5):20–25, September 2008.

[20] Srikanth Kandula et al. Dynamic load balancing without packet reordering. *SIGCOMM Comput. Commun. Rev.*, 2007.

[21] Rishi Kapoor et al. Bullet trains: A study of nic burst behavior at microsecond timescales. CoNEXT '13, New York, NY, USA.

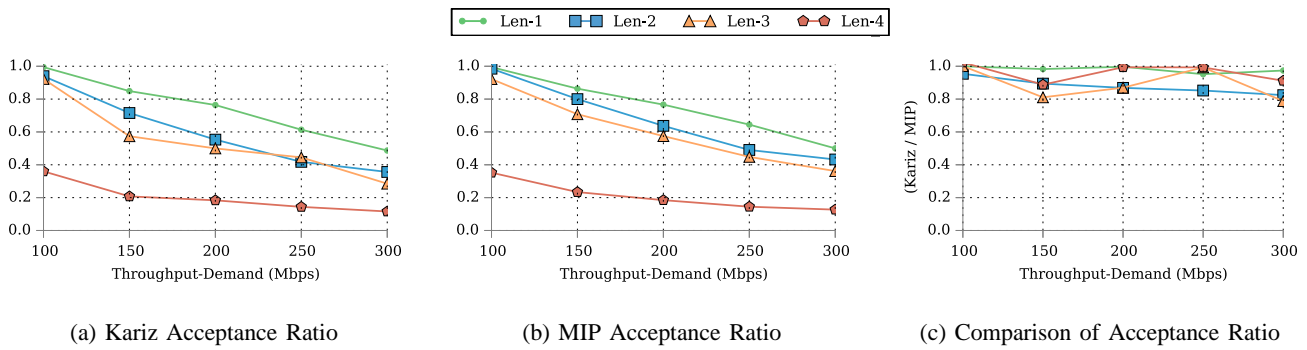[22] Edward Yu-Hsien Lin. A biblographical survey on some wellknown non-standard knapsack problems. 1998.
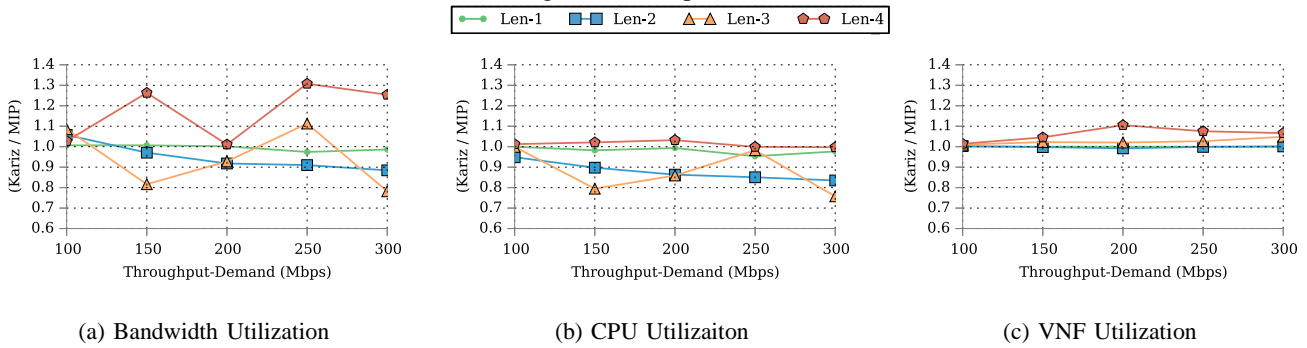
(a) Kariz Acceptance Ratio (b) MIP Acceptance Ratio (c) Comparison of Acceptance Ratio

Figure 5: Acceptance Ratio

(a) Bandwidth Utilization (b) CPU Utilizaiton (c) VNF Utilization

Figure 6: Comparison of Resource Utilization

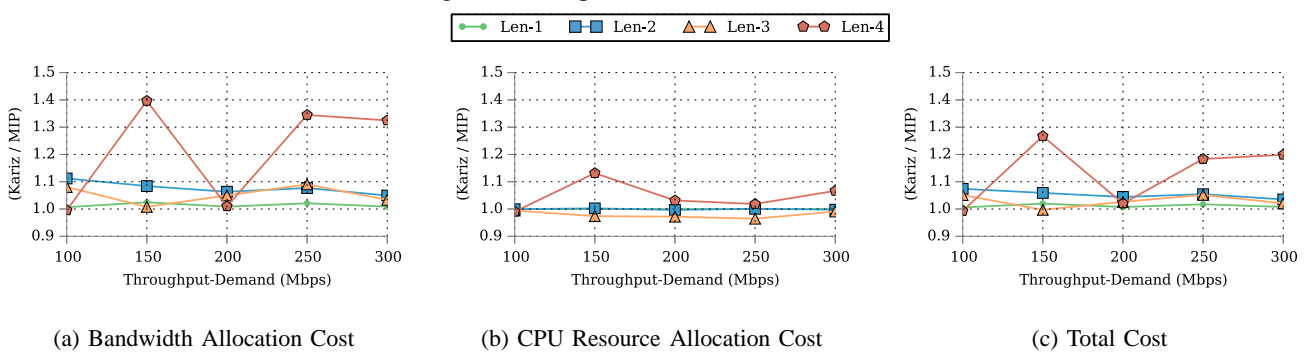(a) Bandwidth Allocation Cost (b) CPU Resource Allocation Cost (c) Total Cost

Figure 7: Operational Costs

[23] Hui Long et al. Laberio: Dynamic load-balanced routing in openflow-enabled networks. In *IEEE AINA, 2013*, 2013.

[24] M. Mahalingam et al. Virtual extensible local area network: A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, 2014.

[25] Mohammad Mahdian and Martin Pál. Universal facility location. In *Algorithms-ESA 2003*, pages 409–421. Springer, 2003.

[26] Joao Martins et al. Clickos and the art of network function virtualization. In *NSDI 2014*.

[27] H. Moens and F. D. Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *IEEE CNSM '14*.

[28] Martin Pal et al. Facility location with nonuniform hard capacities. In *Proceedings. 42nd IEEE Symposium on Foundations of Computer Science, 2001*.

[29] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, December 1999.

[30] P. Quinn and T. Nadeau. Service function chaining problem statement. Technical report, 2014.

[31] Shriram Rajagopalan et al. Split/merge: System support for elastic execution in virtual middleboxes. In *NSDI 2013*.

[32] S. Sahhaf et al. Network service chaining with efficient network function mapping based on service decompositions. In *NetSoft, 2015*.

[33] Derek L. Schuff et al. Conservative vs. optimistic parallelization of stateful network intrusion detection. In *12th ACM SIGPLAN*, 2007.

[34] V. Sekar et al. Design and implementation of a consolidated middlebox architecture. In *Proc. of USENIX NSDI '12*.

[35] Shan Sinha et al. Harnessing TCPs Burstiness using Flowlet Switching. In *HotNets*, San Diego, CA, November 2004.

[36] Robin Sommer et al. Hilti: An abstract execution environment for deep, stateful network traffic analysis. In *IMC 2014*.

[37] JA Tomlin. Minimum-cost multicommodity network flows. *Operations Research*, 14(1):45–51, 1966.

[38] Javier Verdú et al. Multilayer processing - an execution model for parallel stateful packet processing. In *ANCS 2008*.

[39] Minlan Yu et al. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM 2008*.